# **1 - Tri Rapide (Quick Sort)**

## **1.1. Définition**

Le **tri rapide** est un **algorithme de tri efficace** basé sur la méthode **« diviser pour régner »**.  
 Il fonctionne en **choisissant un élément pivot**, puis en **réarrangeant le tableau** pour que tous les éléments inférieurs au pivot soient à gauche et tous les éléments supérieurs soient à droite.  
 Le processus est ensuite appliqué **récursivement** aux sous-tableaux gauche et droit.

## **1.2. Étapes principales de l’algorithme**

1. **Choisir un pivot**

L’élément pivot peut être choisi de différentes façons : le premier élément, le dernier, ou un élément aléatoire.

1. **Partitionner le tableau**
   * Réorganiser les éléments pour que :
     + Les éléments plus petits que le pivot soient à gauche
     + Les éléments plus grands que le pivot soient à droite
2. **Appliquer récursivement le tri**
   * Trier récursivement le sous-tableau gauche
   * Trier récursivement le sous-tableau droit
3. **Combiner les résultats**
   * Comme le tableau est trié sur place, aucune étape de combinaison n’est nécessaire.

## **1.3. Exemple**

Supposons le tableau :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **8** | **3** | **7** | **6** | **2** | **5** |

### **Étape 1 : Choisir le pivot**

* Pivot = 5 (dernier élément)

### **Étape 2 : Partitionner**

* Déplacer les éléments inférieurs à 5 à gauche et les supérieurs à droite :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **3** | **2** | **5** | **6** | **8** | **7** |

**Étape 3 : Trier récursivement**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sous-tableau gauche : | **3** | **2** | —-> Pivot = 2 —-> | **2** | **3** |

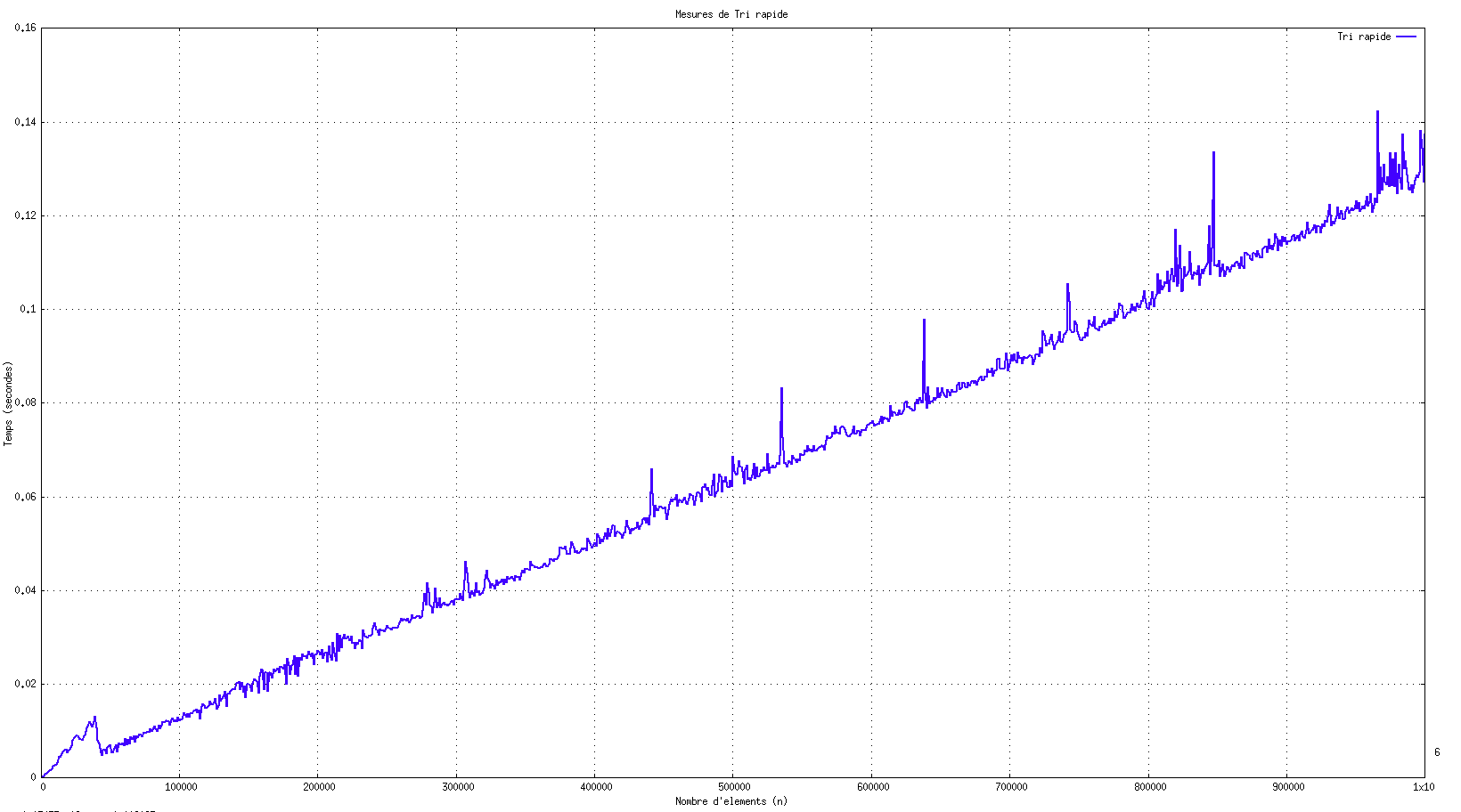
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Sous-tableau droit : | **6** | **8** | **7** | —> Pivot = 7 —> | **6** | **7** | **8** |

**Résultat final :**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **2** | **3** | **5** | **6** | **7** | **8** |

# **1.4.Visualisation des temps d’exécution des algorithmes de tri rapide**

# Tester le tri rapide sur un tableau contenant des tailles différentes, de 1000 à 1 000 000.



**2 - Tri par fusion (Merge Sort)**

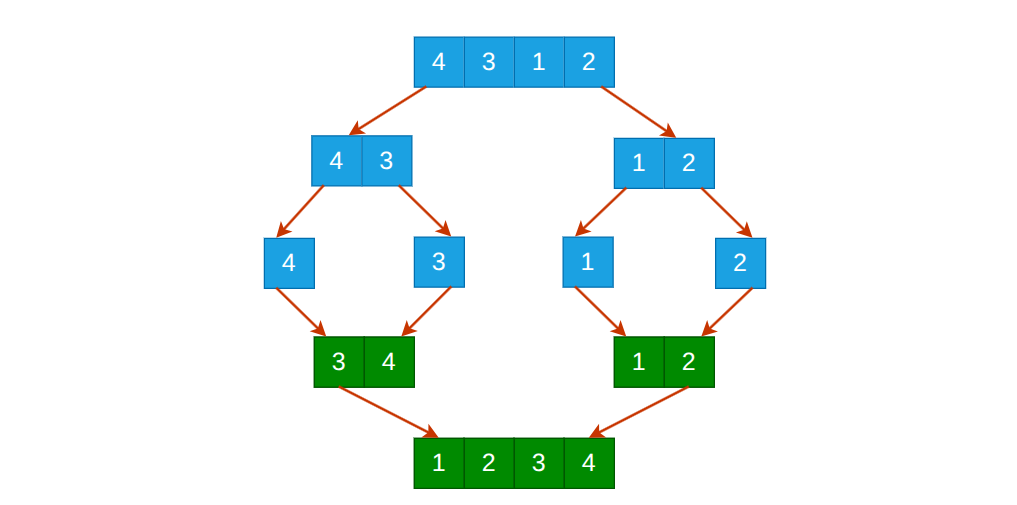
## **2.1. Définition**

Le **tri par fusion** est un **algorithme de tri efficace** basé sur la méthode **« diviser pour régner »**. Il fonctionne en **divisant le tableau en sous-tableaux plus petits**, en **triant chaque sous-tableau**, puis en **fusionnant les sous-tableaux triés** pour obtenir un tableau final trié.

## **2.2. Étapes principales de l’algorithme**

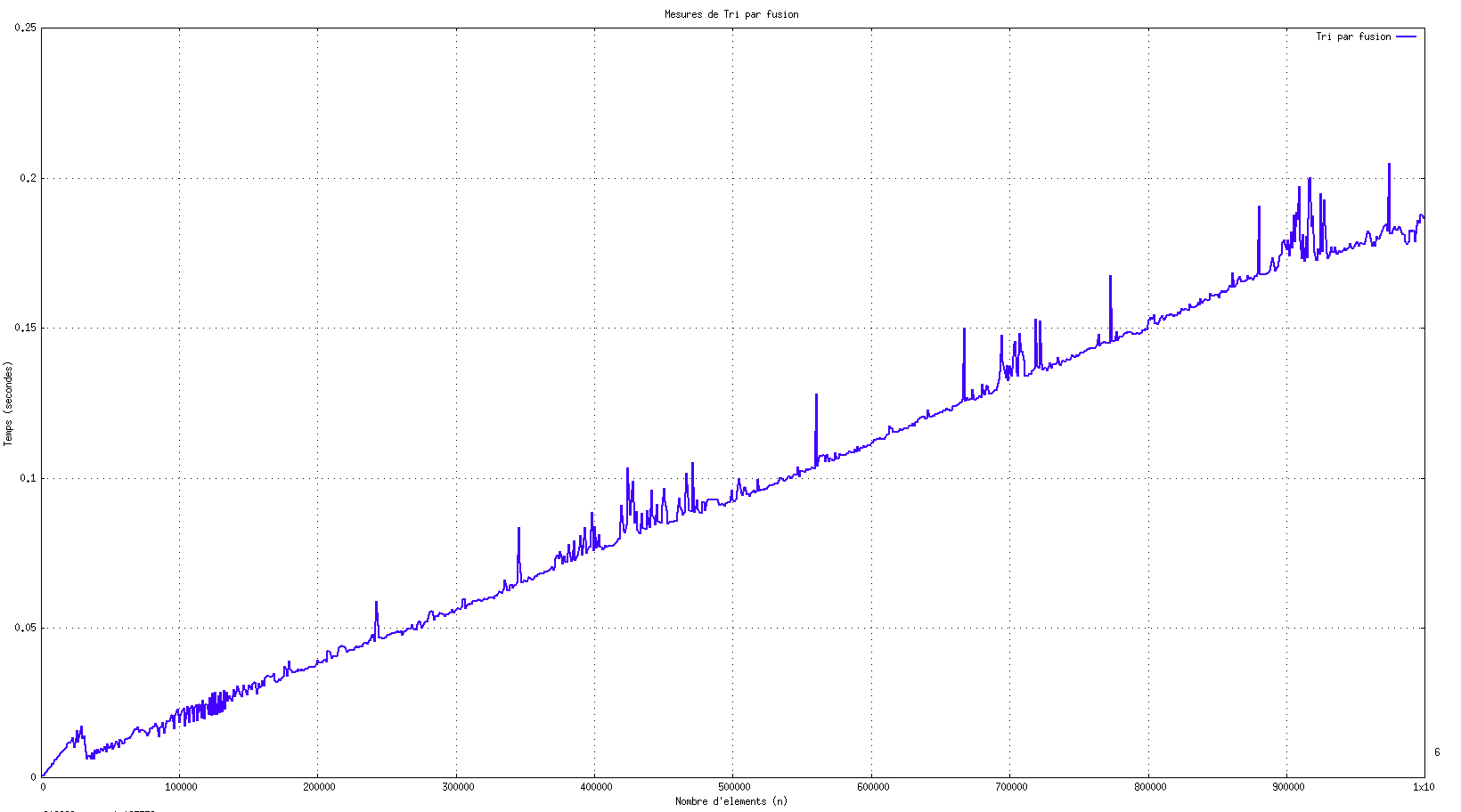
1. **Diviser le tableau**
   * Diviser le tableau en deux moitiés de taille approximativement égale.
2. **Trier récursivement chaque moitié:** Appliquer récursivement le tri par fusion sur la moitié gauche et sur la moitié droite.
3. **Fusionner les sous-tableaux triés**
   * Combiner les deux sous-tableaux triés pour former un seul tableau trié.
   * Cette étape s’appelle **merge** (fusion).

## **2.3. Exemple**



## **2.4. Visualisation des temps d’exécution des algorithmes de tri par fusion**

# Tester le tri rapide sur un tableau contenant des tailles différentes, de 1000 à 1 000 000.



# **3 - Tri par tas (Heap Sort)**

## **3.1. Définition**

Le **tri par tas** (*Heap Sort*) est un algorithme de tri qui transforme un tableau en un **tas maximum** (max-heap).  
 Dans un **max-heap**, chaque parent est **plus grand** que ses enfants.

Pour trier un tableau en ordre croissant :

1. On construit un **tas maximum**.
2. On échange la racine (le maximum) avec le dernier élément du tableau.
3. On réduit la taille du tas.
4. On répare le tas avec **heapify down**.
5. On répète jusqu'à ce que tout soit trié.

## **3.2. Fonction heapify down**

## **Heapify down = descente dans le tas**

Elle part d’un index donné et fait descendre l’élément :

* Comparez le parent avec ses deux enfants.
* Échange avec le **plus grand enfant** si nécessaire.
* Continue jusqu’à ce que la propriété du tas soit respectée.

## **3.3. Les étapes du tri par tas**

### **Étape 1 : Construire le tas (Heapify / Build Max-Heap)**

On transforme le tableau initial en un **tas maximum**.  
 On part du **dernier parent** jusqu'à l’indice 0 et on applique la procédure heapify.

### **Étape 2 : Extraire le maximum**

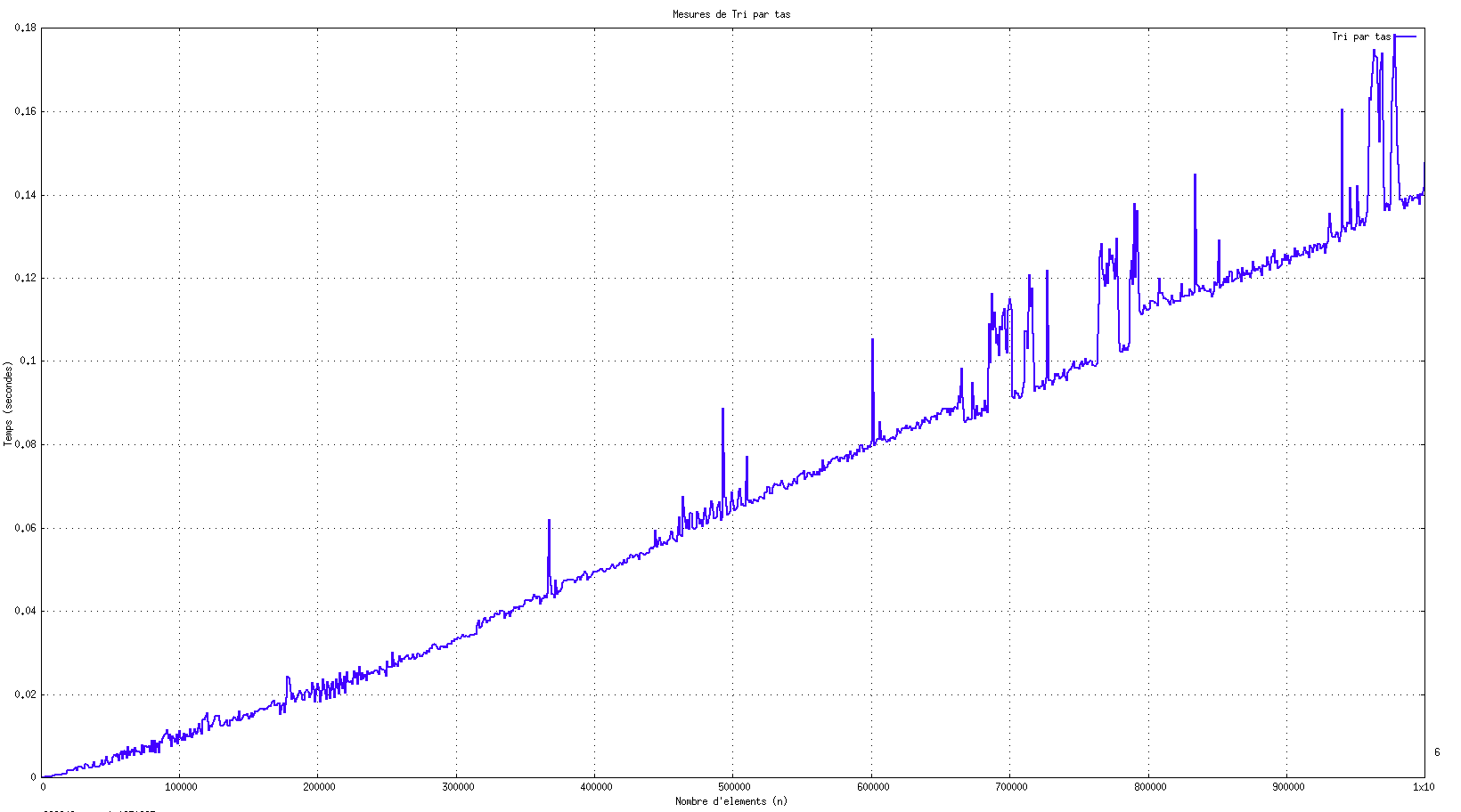
1. La racine du tas (indice 0) est le **plus grand élément**.
2. On l’échange avec l’élément à la fin du tableau.
3. On réduit la taille du tas (on ignore la dernière position, qui est maintenant bien placée).
4. On ré-applique heapify down sur le premier élément dans le tableau pour restaurer le tas.

### **Étape 3 : Répéter**

On répète l’extraction du maximum jusqu’à ce que tous les éléments soient placés.

## **3.4. Visualisation des temps d’exécution des algorithmes de tri par tas**

## Tester le tri rapide sur un tableau contenant des tailles différentes, de 1000 à 1 000 000.



# **4 - Tri par Bulle (Bubble Sort )**

## **4.1. Définition**

L’algorithme de tri à bulles (*Bubble Sort*) est le plus simple des algorithmes de tri.  
Il fonctionne en **échangant à plusieurs reprises les éléments adjacents** lorsqu’ils ne sont pas dans le bon ordre.

Il est souvent utilisé pour introduire le concept de tri, et il est particulièrement adapté pour trier de **petits ensembles de données**.

* **Time Complexity:** O(n2)
* **Space Complexity:** O (1)

## **4.2. Les étapes du tri par BULLE**

Pour trier un ensemble de données en utilisant le tri à bulles, suivez les étapes ci-dessous :

1. Commencez par comparer les deux premiers éléments , S’ils ne sont pas dans le bon ordre, échangez-les.
2. Continuez ce processus pour tous les éléments en avançant de gauche à droite.  
   Après le premier passage, le plus grand élément se trouvera à la fin.
3. Lors du passage suivant, ignorez le dernier élément car il est déjà trié, puis répétez les étapes précédentes.  
   Le deuxième plus grand élément atteindra la deuxième dernière position.
4. Répétez ces étapes jusqu’à ce que tout le tableau soit trié.

Cet algorithme trie les éléments dans l’ordre croissant, mais avec quelques modifications, il peut trier dans n’importe quel ordre souhaité.

## **4.4. Visualisation des temps d’exécution des algorithmes de tri par BULL**

// Ajouter graph ici

# **5 - Tri par SHELL (Shell Sort )**

## **5.1. Définition**

Le **Shell Sort**, également appelé *méthode de Shell*, est un algorithme de tri **en place** basé sur la comparaison.  
C’est une optimisation du **tri par insertion (Insertion Sort)**.

Il améliore l'efficacité du tri par insertion en permettant de déplacer des éléments sur de **plus grandes distances** lors des premières étapes, ce qui réduit considérablement le nombre d’échanges nécessaires, surtout pour les grands ensembles de données.

Le Shell Sort a été inventé par **Donald Shell** en **1959**, et il est considéré comme le premier algorithme à avoir brisé la **barrière de complexité en O(n²)** pour les algorithmes de tri.

## **5.2. Les étapes du tri par BULLE**

Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect. Une image contenant texte, capture d’écran, nombre, Police

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, nombre, Police

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.

# **6- Conclusion**

Le tri Bulle est très simple à comprendre et à implémenter, mais il devient extrêmement lent dès que la taille du tableau augmente. Pour 100 000 éléments, son temps d’exécution dépasse 30 secondes, ce qui le rend inadapté aux grandes données.

Le tri Shell améliore considérablement le tri Bulle en réduisant les inversions lointaines. Il reste très rapide même pour des tableaux de taille moyenne à grande, avec un temps de quelques millisecondes pour 100 000 éléments.

Le tri Fusion est stable et sa complexité est toujours . Il offre des performances fiables et prévisibles, avec des temps d’exécution très courts même pour de grandes tailles.

Le tri Rapide est généralement le plus rapide en pratique pour des tableaux en mémoire, avec un temps très faible pour de grandes tailles. Sa complexité moyenne est , mais le pire cas reste rare si le pivot est bien choisi.

Le tri par Tas a également une complexité et reste stable en performance, mais il est légèrement plus lent que le tri Rapide. Il est fiable pour les grandes données, même si son implémentation est un peu plus complexe et qu’il n’est pas stable.

En résumé, pour de petites tailles ou à des fins pédagogiques, le tri Bulle peut suffire. Pour des tailles moyennes, le tri Shell est efficace. Pour de grandes tailles, les tris Rapide et Fusion sont les meilleurs choix, tandis que le tri par Tas est une alternative stable et fiable.

Une image contenant texte, ligne, Tracé, capture d’écran

Le contenu généré par l’IA peut être incorrect.